**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе № 2**

**по дисциплине «Программирование»**

Тема: «**ОДНОМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3373 |  | Огородникова В.С. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Изучение и организация структур; получение практических навыков работы со структурами; определение преимуществ и недостатков использования структур.

**Основные теоретические положения.**

***Объявление одномерных массивов***

Объявление в программах одномерных массивов выполняется в соответствии со следующим правилом:

<Базовый тип элементов> <Идентификатор массива> [<Количество элементов>]

Например:

int ArrInt [10], A1 [20];

double D [100];

char Chars [50];

bool B [200];

 Значения индексов элементов массивов всегда начинается с 0. Поэтому максимальное значение индекса элемента в массиве всегда на единицу меньше количества элементов в массиве.

Обращение к определенному элементу массива осуществляется с помощью указания значения индекса этого элемента:

A1 [8] = -2000;

cout << A1 [8];    // На экран выведено -2000

В этом примере, обратившись к элементу массива A1 с индексом 8, мы, фактически, обратились к его 9-му элементу.

При обращении к конкретному элементу массива этот элемент можно рассматривать как обычную переменную, тип которой соответствует базовому типу элементов массива, и осуществлять со значением этого элемента любые операции, которые характерны для базового типа. Например, поскольку базовым типом массива A1 является тип данных int, с любым элементом этого массива можно выполнять любые операции, которые можно выполнять над значениями типа int.

При объявлении массива его можно инициализировать определенными значениями:

short S [5] = {1, 4, 9, 16, 25};

или так:

short S [ ] = {1, 4, 9, 16, 25};

Во втором случае мы не указываем количество элементов массива S. Автоматически создается массив на 5 элементов в соответствии с инициализирующими значениями.

Эти инициализации будут эквивалентны следующим операциям присваивания:

S[0] = 1;

S[1] = 4;

S[2] = 9;

S[3] = 16;

S[4] = 25;

Количество значений, указанных в фигурных скобках (инициализирующих значений) не должно превышать количества элементов в массиве (в нашем примере - 5).

Значения всех элементов массива в памяти располагаются в непрерывной области одно за другим. Общий объем памяти, выделяемый компилятором для массива, определяется как произведение объема одного элемента массива на количество элементов в массиве и равно:

sizeof( <Базовый тип> )  \* <Количество элементов>

Для предыдущего примера объем массива S будет равен sizeof( short) \* 5 = 2 \* 5 = 10 байтам.

Поскольку все элементы массивов располагаются в памяти один за другим без разрывов, обращение к элементам массива по их индексам (какой бы длины не был этот массив) осуществляется очень эффективно путем вычисления адреса нужного элемента. Пусть, например, адрес памяти, где начинается массив  S, равен 100, тогда адрес элемента этого массива с индексом 3 будет равен 100 + sizeof( short) \* 3 = 100 + 2 \* 3 = 106. Обращаемся по этому адресу и считываем 2 байта. Это и будет значением элемента с индексом 3 массива S.

В языке C++ не осуществляется проверка выхода за границы массивов. То есть, вполне корректно (с точки зрения компилятора) будет обращение к элементу массива S, индекс которого равен 10. Это может привести к возникновению весьма серьезных отрицательных последствий. Например, если выполнить присвоение S[10] = 1000 будут изменены данные, находящиеся за пределами массива, а это может быть значение какой-нибудь другой переменной программы. После этого предсказать поведение программы будет невозможно. Единственный выход – быть предельно внимательным при работе с индексами элементов массивов.

***Объявление многомерных  массивов***

Многомерные массивы определяются аналогично одномерным массивам. Количество элементов по каждому измерению указывается отдельно в квадратных скобках:

int  A1 [5] [3];                 //  Двумерный массив, элементами которого являются

                                         //   значения типа int

double D [10] [15] [3];   //  Трехмерный массив, элементами которого являются

                                         //   значения типа double

Здесь массив **A1**представляет собой обычную двумерную матрицу из 5-ти строк и 3–х столбцов.

Массив **D** – трехмерный массив, который можно представить как трехмерный параллелограмм из 3-х двумерных матриц.

Общее число элементов в многомерном массиве определяется как произведение количества элементов по каждому измерению. Так, например, массив **D** содержит **10 \* 15 \* 3 = 450**  элементов типа **double**, а объем памяти, требующийся для этого массива, будет равен **450 \* 8 = 3600** байтам.

Массивы с большим, чем 3, количеством измерений используются достаточно редко. Одной из причин этого является быстрый рост объема памяти, необходимой для размещения таких массивов.

В следующей таблице показана схема размещения элементов массива **A1** в памяти:



Так же как и в одномерном массиве, элементы многомерных массивов располагаются друг за другом в непрерывном участке памяти.

При определении многомерные массивы могут инициализироваться определенными значениями. Для получения массива **A1**с теми значениями элементов, которые приведены в таблице, можно инициализировать массив следующим образом:

int  A1 [5] [3] =

     {

          1, 1, 1,

          2, 4, 8,

          3, 9, 27,

          4, 16, 64,

          5, 25, 125

      };

или так:

int  A1 [ ] [3] =

     {

          1, 1, 1,

          2, 4, 8,

          3, 9, 27,

          4, 16, 64,

          5, 25, 125

      };

Во втором определении опущен размер массива по первому индексу, при этом автоматически ему назначается значение 5 в соответствии с инициализирующими значениями. Размеры массивов по второму и по следующим индексам опускать нельзя.

Для доступа к определенному элементу многомерного массива необходимо указать в квадратных скобках конкретные значения всех индексов этого элемента. Например:

cout << A1 [1] [2];     // На экран выведено значение 8

**Алгоритм бинарного поиска на С++:**

1. Определить функцию для бинарного поиска, которая принимает отсортированный массив, значение, которое нужно найти, и индексы начала и конца подмассива, в котором происходит поиск.

int binarySearch(int arr[], int value, int start, int end) {

    if (end >= start) {

        int mid = start + (end - start) / 2;

        if (arr[mid] == value) {

            return mid;

        }

        if (arr[mid] > value) {

            return binarySearch(arr, value, start, mid - 1);

        }

        return binarySearch(arr, value, mid + 1, end);

    }

    return -1;

}

2. В главной функции инициализировать отсортированный массив и значение, которое нужно найти.

int main() {

    int arr[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

    int value = 10;

    int size = sizeof(arr) / sizeof(arr[0]);

    int result = binarySearch(arr, value, 0, size - 1);

    if (result == -1) {

        cout << "Элемент не найден" << endl;

    } else {

        cout << "Элемент находится в позиции " << result << endl;

    }

    return 0;

}

Этот алгоритм сначала проверяет, находится ли значение в середине отсортированного массива.

* Если да, то возвращает индекс этого значения.
* Если значение меньше середины массива, то алгоритм рекурсивно вызывает себя для левой половины массива.
* Если значение больше середины массива, то алгоритм рекурсивно вызывает себя для правой половины массива.
* Если значение не найдено, то функция возвращает -1.

**Постановка задачи.**

Разработать алгоритм и написать программу, которая позволяет:

1)    Создает целочисленный массив размерности *N* = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.

2)    Отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.

3)    Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.

4)    Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитайте время поиска.

5)    Выводит количество элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.

6)    Выводит количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

7)    Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором. (\*)

8)     Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

Должна присутствовать возможность запуска каждого пункта многократно.

Таблица 1 – Варианты сортировок

|  |  |
| --- | --- |
| Номер варианта | Название сортировки |
| 1 | Bubble sort (пузырьковая сортировка) |
| 2 | Shaker sort (шейкер-сортировка) |
| 3 | Comb sort (сортировка расчёской) |
| 4 | Insert sort (сортировка вставками) |

**Выполнение работы.**

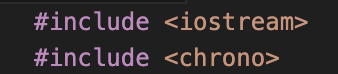


Рис.1. – подключение библиотек

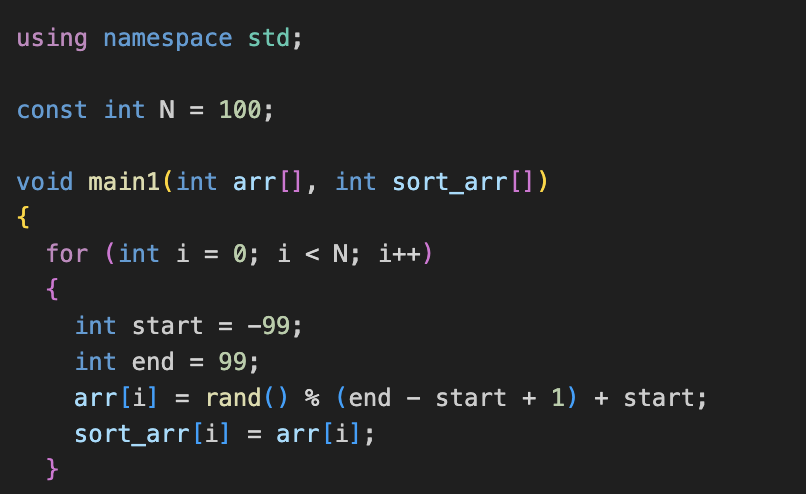


Рис.2. – создаем целочисленный массив размерности N = 100.

Затем производим сортировки и определяем время, затраченное на сортировку, все по пунктам кроме 5\* сортировки:

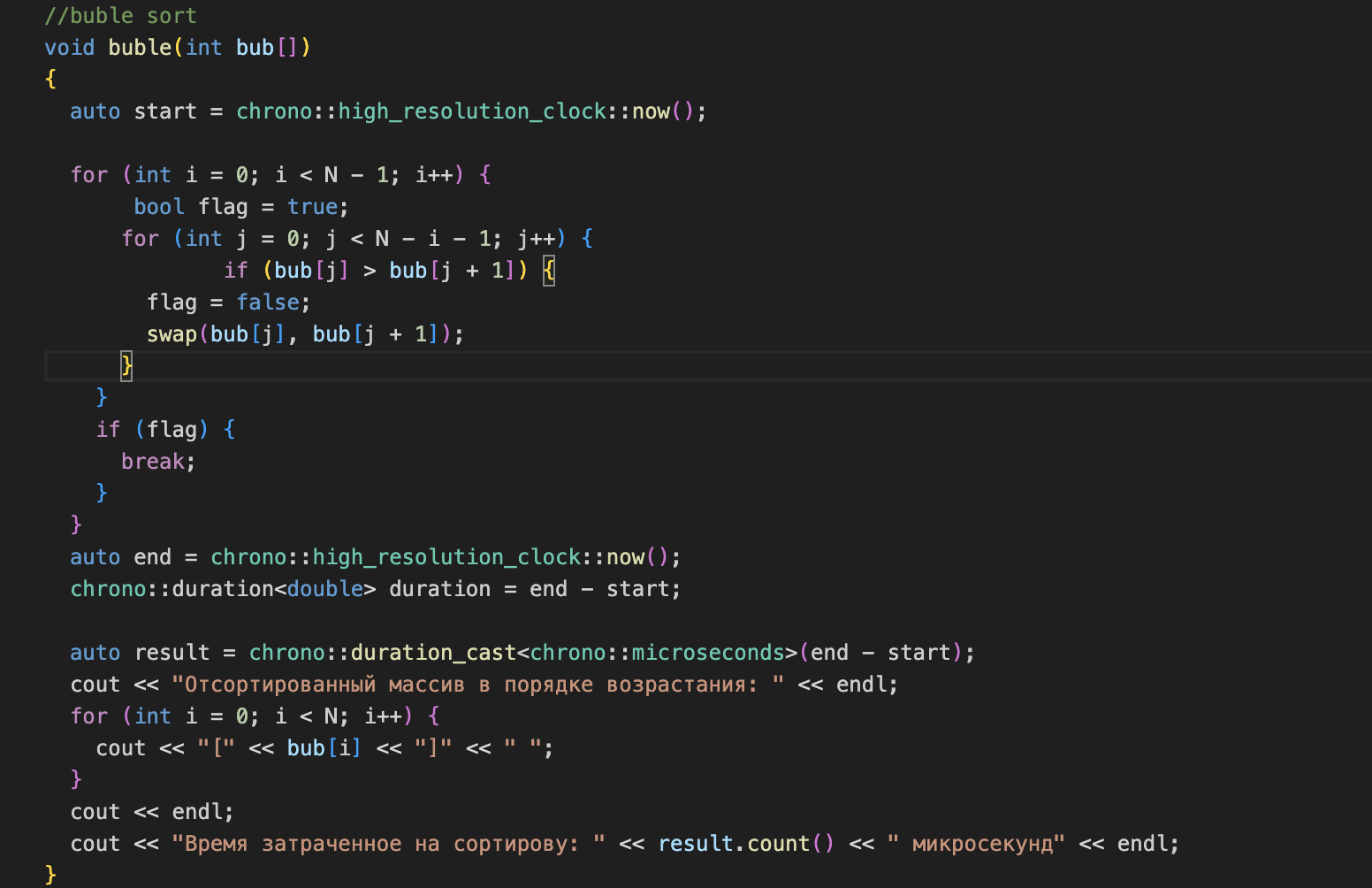


Рис.3. – бабл сорт/пузырьками

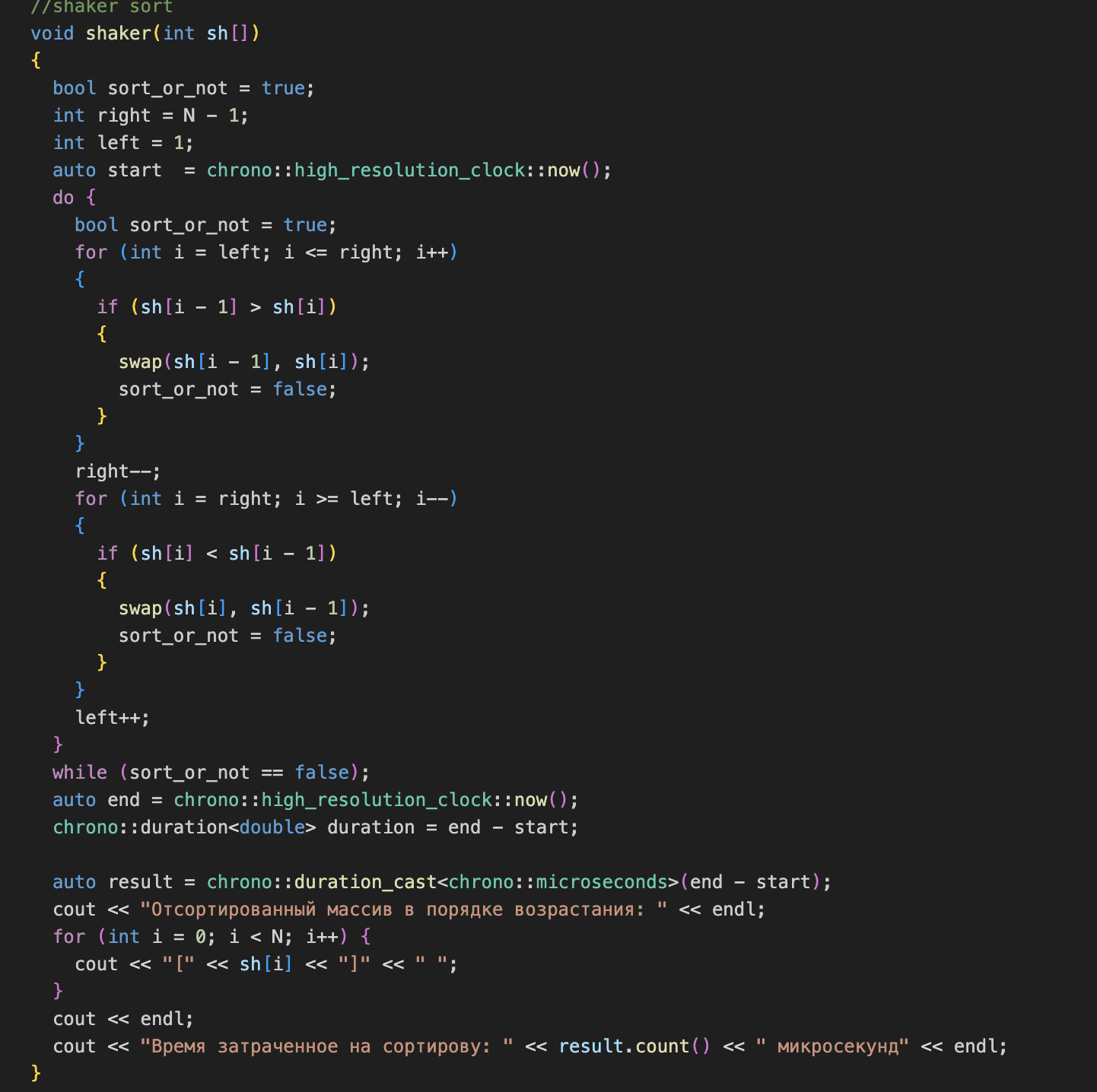


Рис.4. – шейкер сортировка

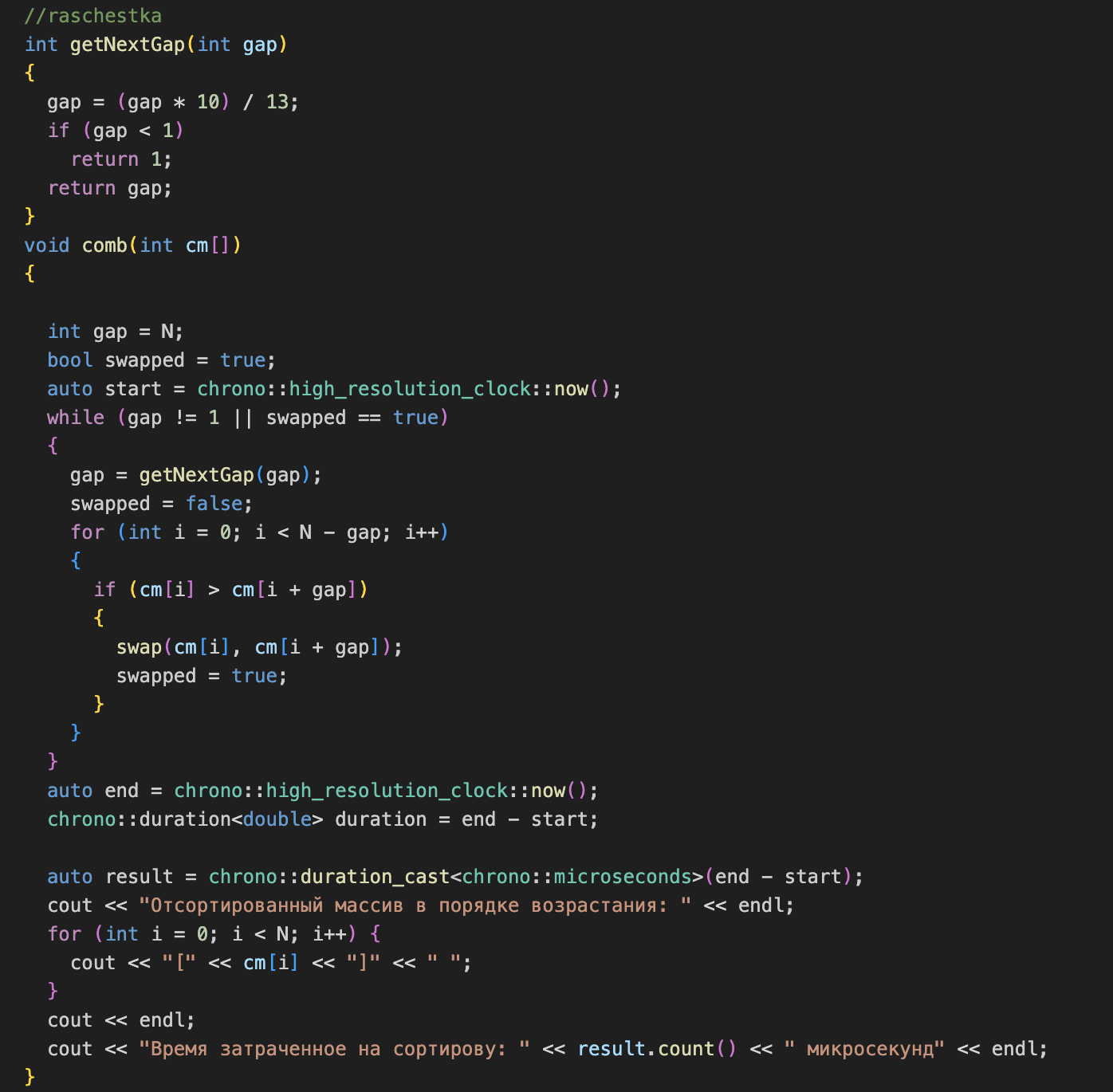


Рис.5. – сортировка расческой



Рис.6. – сортировка вставками

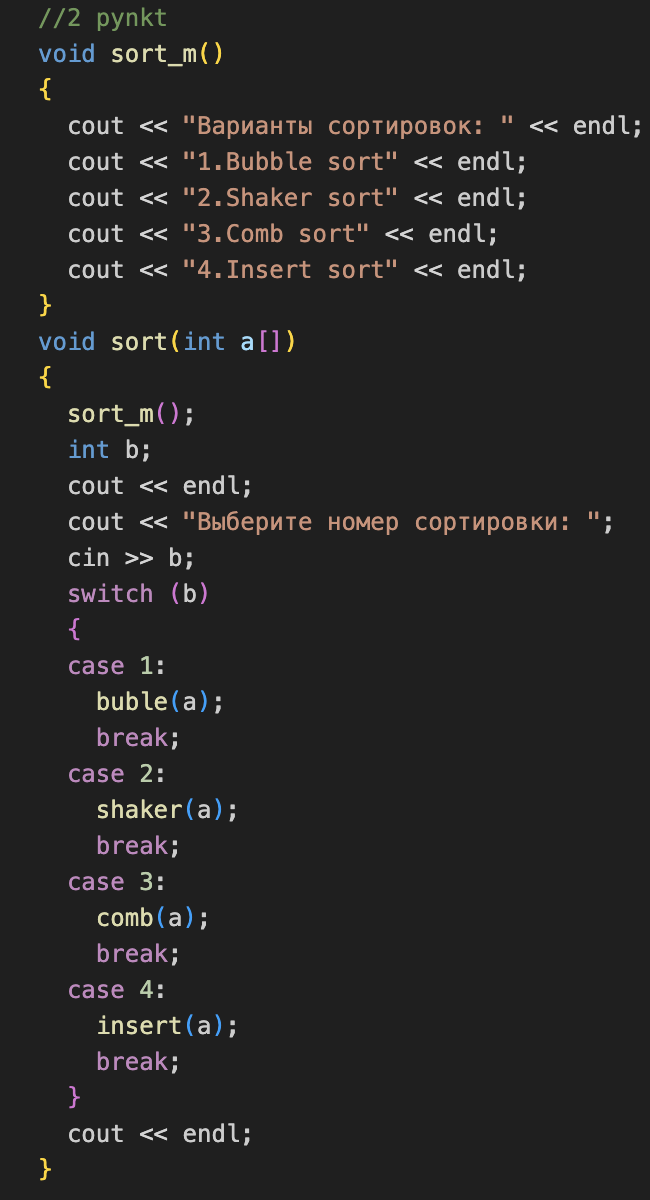
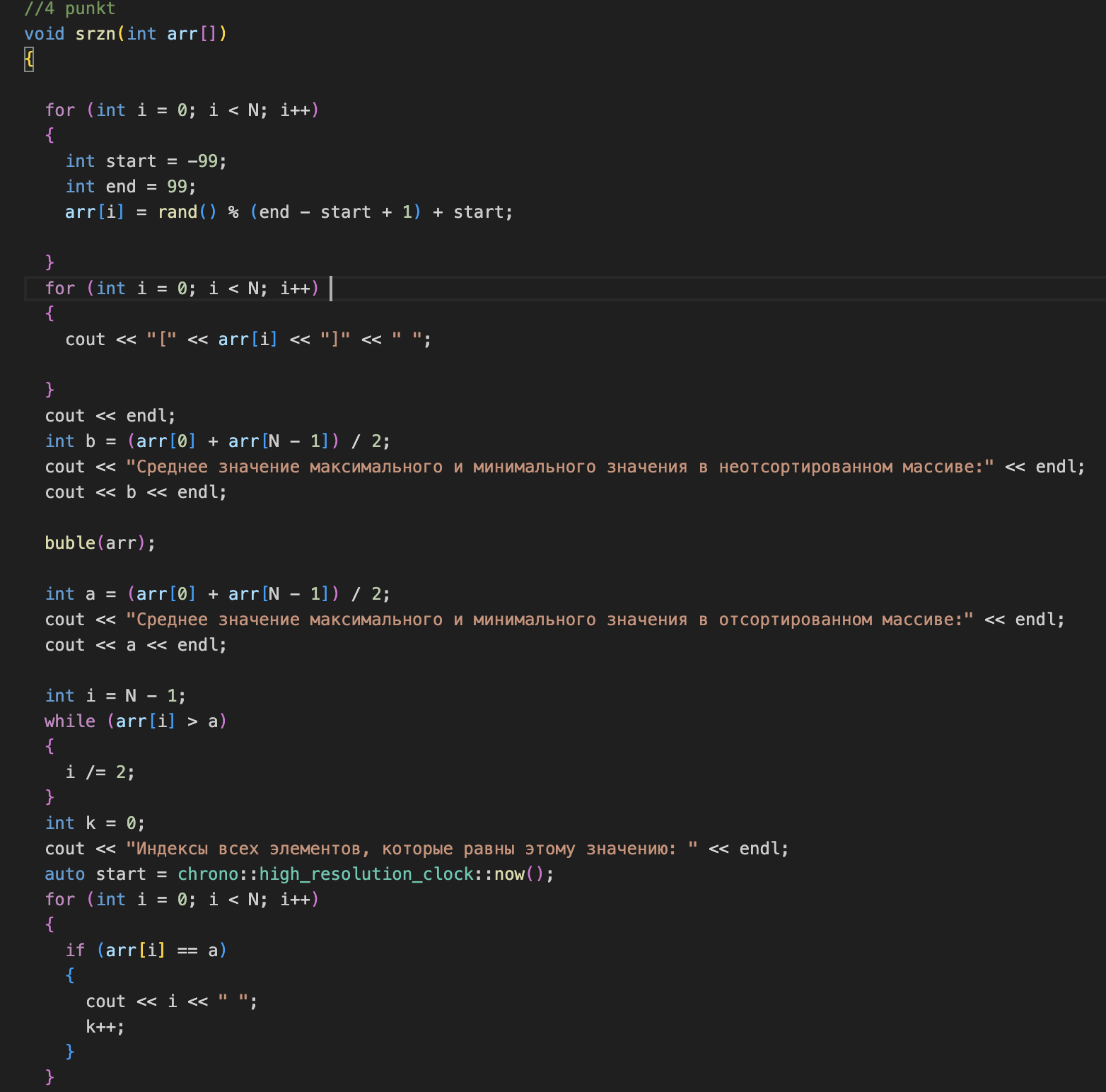


Рис.7. – объединение в меню всех сортировок



Рис.8. – поиск максимального и минимального элемента массива в отсортированном массиве и не в отсортированном



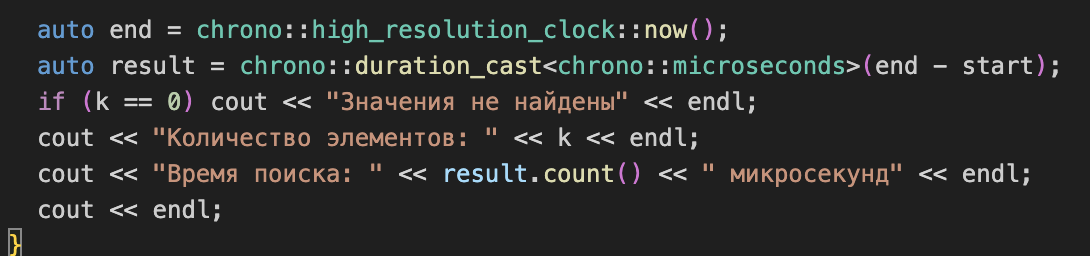


Рис.9. – среднее значение максимального и минимального элемента массива в отсортированном массиве и не в отсортированном

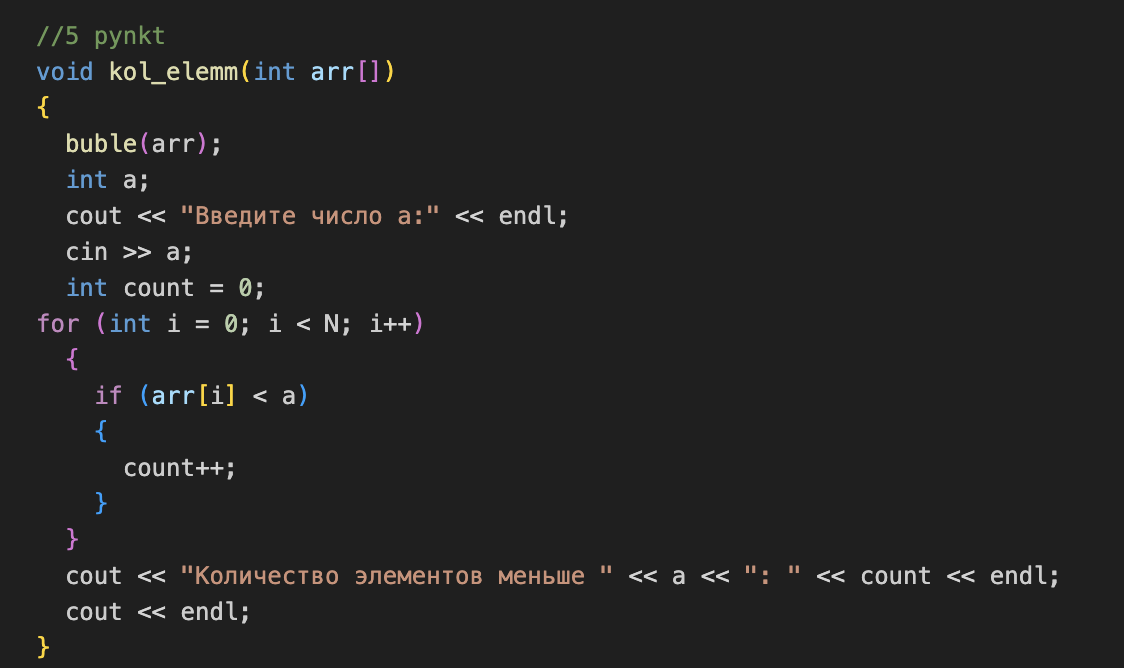


Рис.10. – Вывод количества элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.

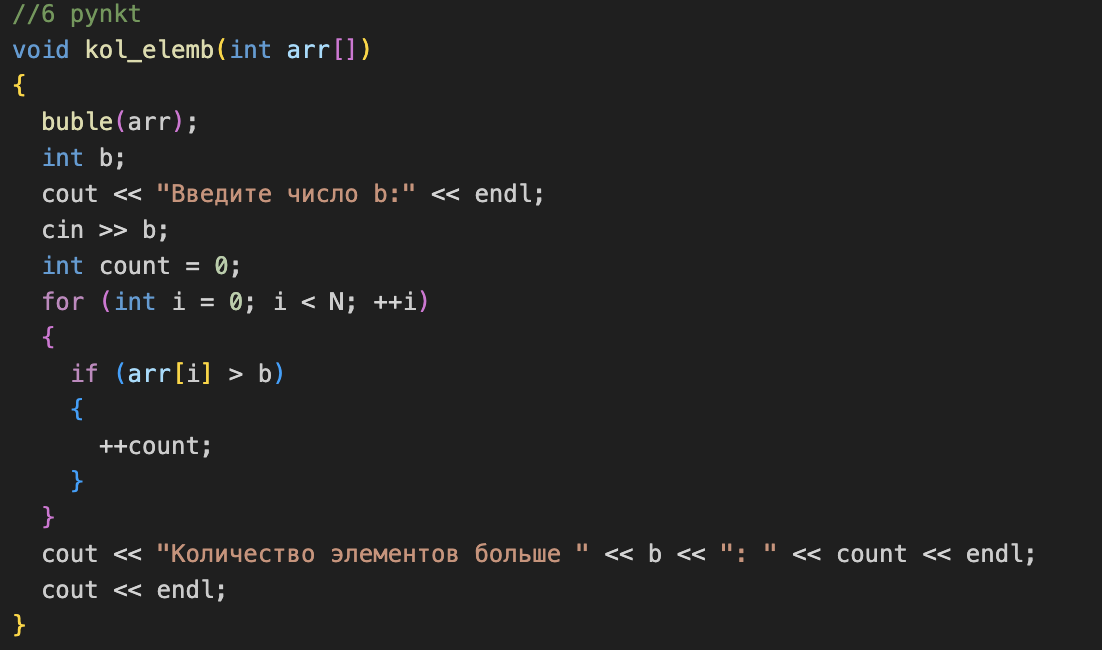


Рис.11. – Вывод количества элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

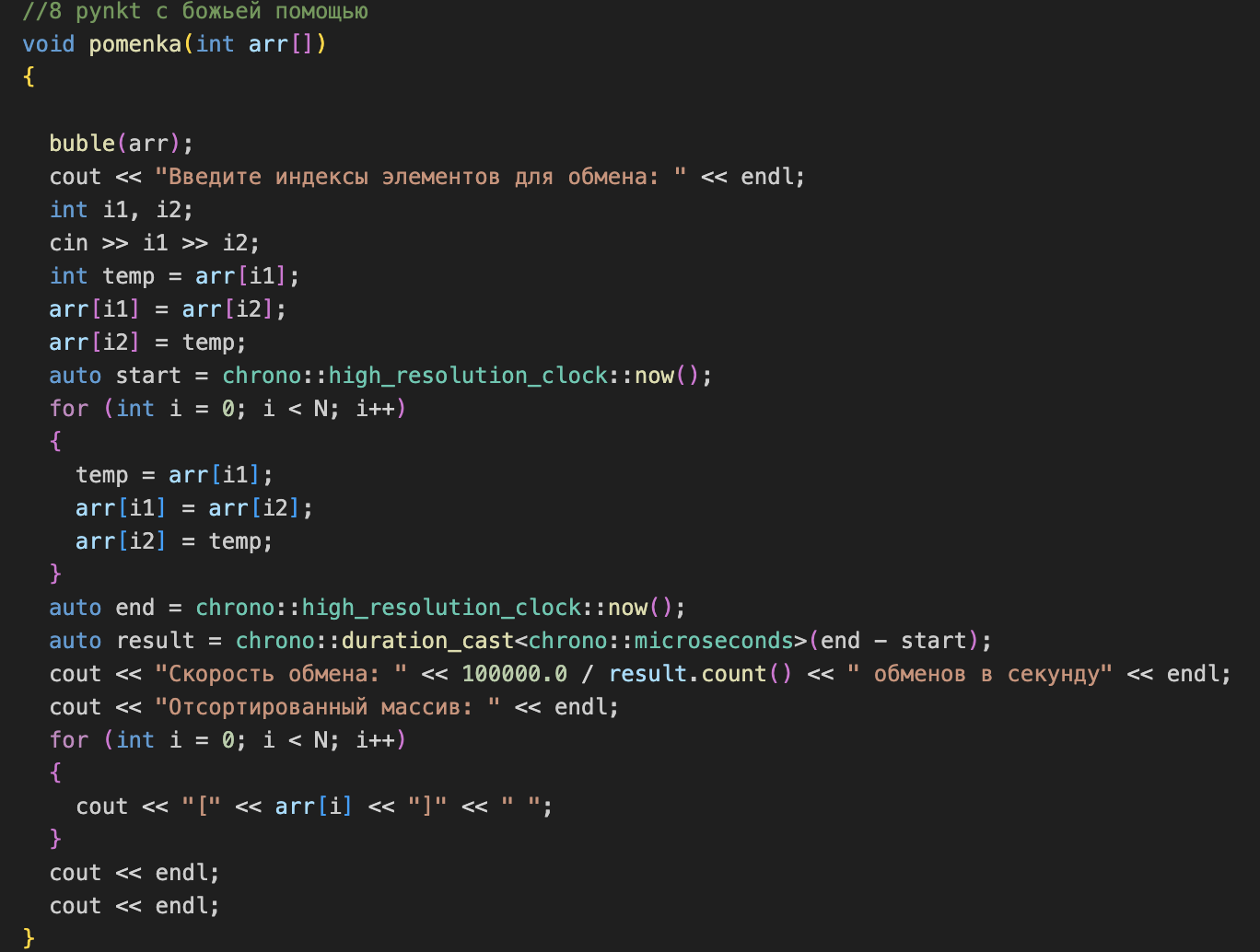


Рис.12. – замена местами элементы массива, индексы которых вводит пользователь

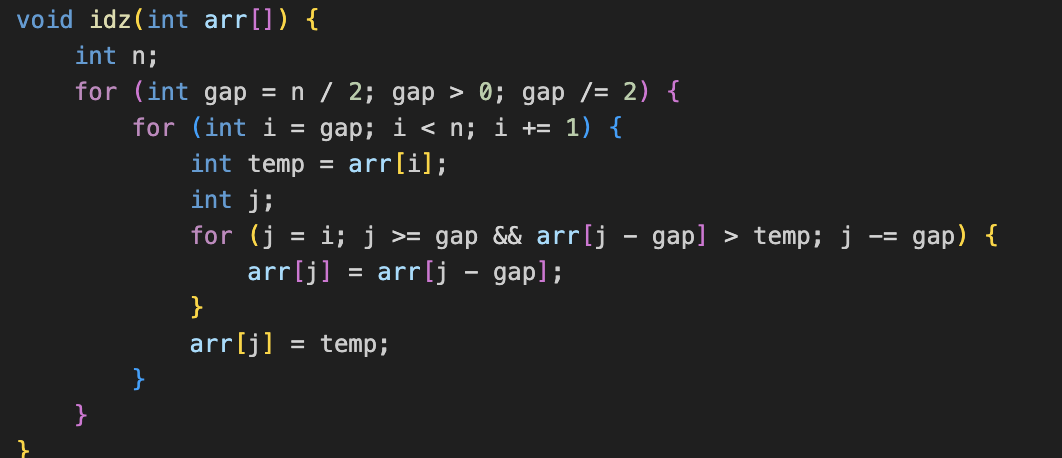
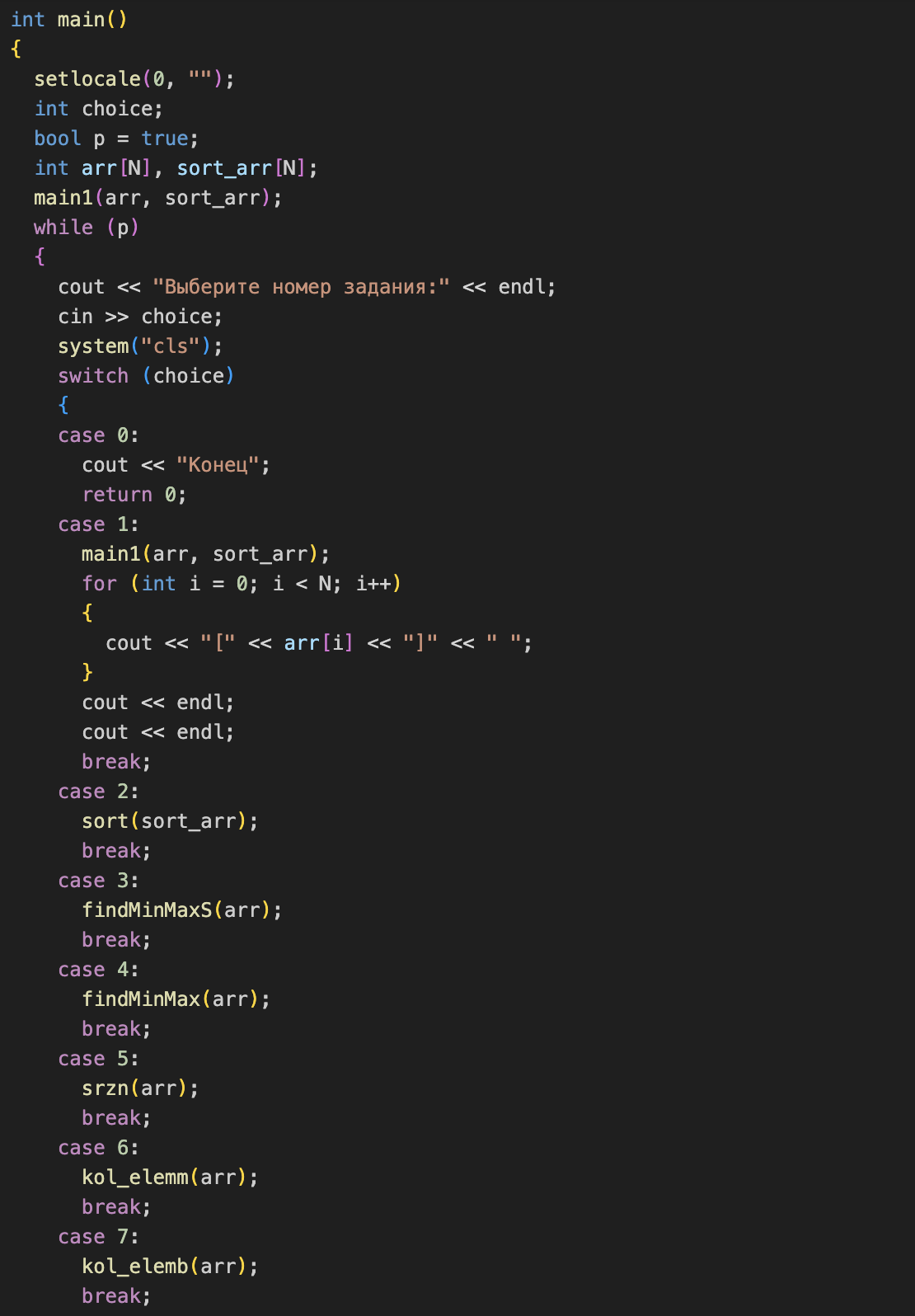


Рис.13. – идз, сортировка Шелла



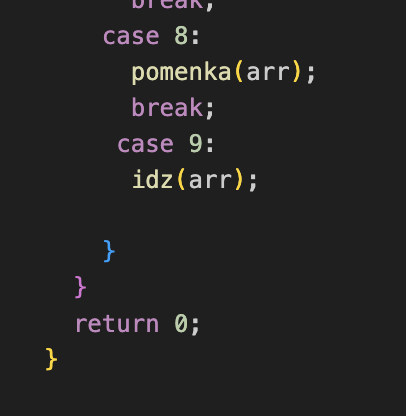


Рис.14. – объединение всех заданий/пунктов в меню

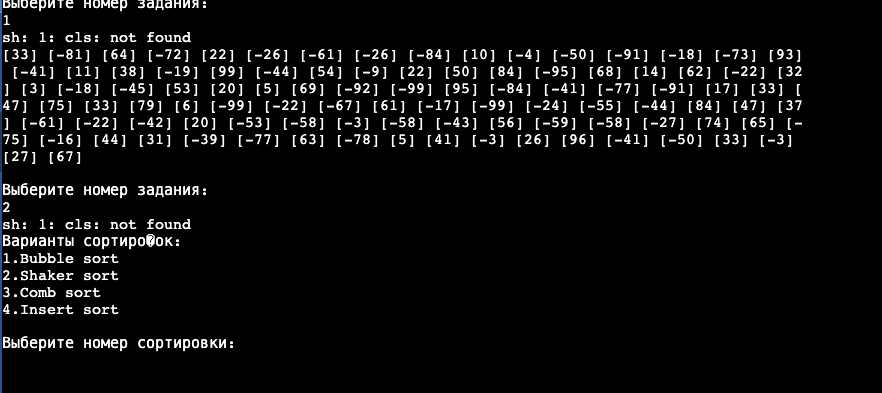


Рис.15. – пример вывода программы

**Вывод.**

Программа работает исправно, с некоторыми недочетами,

Но основные задания имеют рабочий функционал. Не исключайте то, что при копировании кода он просто не заработает, может выдавать ошибки.